CS 476 – Programming Language Design

William Mansky

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Expressions: Big-Step Semantics

$$\frac{(i \text{ is a number literal})}{i \Downarrow i}$$

$$\frac{e_1 \Downarrow i_1 \quad e_2 \Downarrow i_2 \quad (i = i_1 + i_2)}{e_1 \textbf{ + } e_2 \Downarrow i}$$

$$\frac{e_1 \Downarrow i_1 \quad e_2 \Downarrow i_2 \quad (b \text{ is true iff } i_1 \text{ is the same as } i_2)}{e_1 \textbf{ = } e_2 \Downarrow b}$$

$$\frac{e \Downarrow b \quad (\text{if } b \text{ then } e_1 \text{ else } e_2) \Downarrow v}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v}$$

A More Direct Semantic Rule

$$\frac{e \Downarrow \text{true} \quad e_1 \Downarrow v_1}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v_1}$$

$$\texttt{if true then}\ e_1\ \texttt{else}\ e_2 \ \Downarrow\ e_1$$

A More Direct Semantic Rule

$$\frac{e \Downarrow \text{true} \quad e_1 \Downarrow v_1}{\textbf{if}\ e\ \textbf{then}\ e_1\ \textbf{else}\ e_2 \Downarrow v_1}$$

$$\textbf{if true then}\ e_1\ \textbf{else}\ e_2 \rightarrow\ e_1$$

Small-Step Operational Semantics

- Describe how expressions compute, step by step
- $e \rightarrow e'$ means "expression e steps to expression e'"
- Each rule turns an expression into a simpler expression
- A program executes as a sequence of steps:

$$e \rightarrow e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_k \rightarrow v$$

where $e \rightarrow e_1$, $e_1 \rightarrow e_2$, …, $e_k \rightarrow v$ each come from rules

```
(1 + 2) + (3 + 4)  →  3 + (3 + 4)  →  3 + 7 → 10
```

Small-Step Operational Semantics

- Describe how expressions compute, step by step
- $e \to e'$ means "expression e steps to expression e'"
- Each rule turns an expression into a simpler expression
- A program executes as a sequence of steps:

$$e \to e_1 \to e_2 \to \cdots \to e_k \to v$$

where $e \to e_1$, $e_1 \to e_2$, …, $e_k \to v$ each come from rules

- Two kinds of rules: *computation* and *structural*

Expressions: Small-Step Semantics

- Values: int, bool
- Rule 0: Values are done executing!

$$\frac{e_1 \Downarrow i_1 \quad e_2 \Downarrow i_2 \quad (i = i_1 + i_2)}{e_1 \mathbf{+} e_2 \Downarrow i}$$

- Structural rules:

$$\frac{e_1 \rightarrow e_1'}{e_1 \mathbf{+} e_2 \rightarrow e_1' \mathbf{+} e_2} \qquad \frac{e_2 \rightarrow e_2'}{e_1 \mathbf{+} e_2 \rightarrow e_1 \mathbf{+} e_2'}$$

- Computation rule:

$$\frac{(v_1 + v_2 = v)}{v_1 \mathbf{+} v_2 \rightarrow v}$$

Expressions: Small-Step Semantics

- Rule 0: Values are done executing!
- Structural rules: $\dfrac{e_1 \to e_1'}{e_1 \,\mathbf{+}\, e_2 \to e_1' \,\mathbf{+}\, e_2}$ $\qquad$ $\dfrac{e_2 \to e_2'}{e_1 \,\mathbf{+}\, e_2 \to e_1 \,\mathbf{+}\, e_2'}$
- Computation rule: $\dfrac{(v_1 + v_2 = v)}{v_1 \,\mathbf{+}\, v_2 \to v}$

```
(3 + 4) + (5 + 6)
```

Expressions: Small-Step Semantics

- Rule 0: Values are done executing!

- Structural rules: $\dfrac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$ $\qquad\qquad$ $\dfrac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$

- Computation rule: $\dfrac{(v_1 + v_2 = v)}{v_1 + v_2 \to v}$

```
(3 + 4) + (5 + 6)
```

Expressions: Small-Step Semantics

- Rule 0: Values are done executing!

- Structural rules: $\dfrac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$ $\dfrac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$

- Computation rule: $\dfrac{(v_1 + v_2 = v)}{v_1 + v_2 \to v}$

`(3 + 4) + (5 + 6)` → `7 + (5 + 6)`

Expressions: Small-Step Semantics

- Rule 0: Values are done executing!
- Structural rules: $\dfrac{e_1 \to e_1'}{e_1 \mathbf{+} e_2 \to e_1' \mathbf{+} e_2}$ $\qquad$ $\dfrac{e_2 \to e_2'}{e_1 \mathbf{+} e_2 \to e_1 \mathbf{+} e_2'}$
- Computation rule: $\dfrac{(v_1 + v_2 = v)}{v_1 \mathbf{+} v_2 \to v}$

`(3 + 4) + (5 + 6)` $\to$ `7 + (5 + 6)` $\to$ `7 + 11`

Expressions: Small-Step Semantics

- Rule 0: Values are done executing!

- Structural rules: $\dfrac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$ $\dfrac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$

- Computation rule: $\dfrac{(v_1 + v_2 = v)}{v_1 + v_2 \to v}$

$(3 + 4) + (5 + 6) \to 7 + (5 + 6) \to 7 + 11 \to 18$

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Expressions: Small-Step Semantics

- Exercise: Write a structural rule for if-then-else.
- Structural rule(s):

$$\frac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2} \qquad \frac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$$

- Computation rules:

$$\frac{}{\texttt{if true then}\ e_1\ \texttt{else}\ e_2 \to e_1}$$

$$\frac{}{\texttt{if false then}\ e_1\ \texttt{else}\ e_2 \to e_2}$$

Expressions: Small-Step Semantics

- Structural rule(s):

$$\frac{e \to e'}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \to \texttt{if}\ e'\ \texttt{then}\ e_1\ \texttt{else}\ e_2}$$

- Computation rules:

$$\frac{}{\texttt{if true then}\ e_1\ \texttt{else}\ e_2 \to e_1}$$

$$\frac{}{\texttt{if false then}\ e_1\ \texttt{else}\ e_2 \to e_2}$$

Using the Small-Step Semantics

$$\frac{}{\texttt{if 1+2=3 then 2*2 else 7} \rightarrow}$$

Using the Small-Step Semantics

$$\frac{e \to e'}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \to \texttt{if}\ e'\ \texttt{then}\ e_1\ \texttt{else}\ e_2}$$

$$\frac{}{\texttt{if 1+2=3 then 2*2 else 7} \to \texttt{if}\ e'\ \texttt{then 2*2 else 7}}$$

Using the Small-Step Semantics

$$\frac{e \to e'}{\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \to \texttt{if}\ e'\ \texttt{then}\ e_1\ \texttt{else}\ e_2}$$

$$\frac{\texttt{1+2=3} \to e'}{\texttt{if 1+2=3 then 2*2 else 7} \to \texttt{if}\ e'\ \texttt{then 2*2 else 7}}$$

Using the Small-Step Semantics

$$\frac{e_1 \to e_1'}{e_1 = e_2 \to e_1' = e_2}$$

$$\frac{\texttt{1+2=3} \to e'}{\texttt{if 1+2=3 then 2*2 else 7} \to \texttt{if}\ e'\ \texttt{then 2*2 else 7}}$$

Using the Small-Step Semantics

$$\frac{e_1 \rightarrow e_1'}{e_1 = e_2 \rightarrow e_1' = e_2}$$

$$\cfrac{\cfrac{\texttt{1+2} \rightarrow e_1'}{\texttt{1+2=3} \rightarrow e_1'\texttt{=3}}}{\texttt{if 1+2=3 then 2}*\texttt{2 else 7} \rightarrow \texttt{if}\ e'\ \texttt{then 2}*\texttt{2 else 7}}$$

Using the Small-Step Semantics

$$\frac{e_1 \rightarrow e_1'}{e_1 = e_2 \rightarrow e_1' = e_2}$$

$$\dfrac{\dfrac{\texttt{1+2} \rightarrow e_1'}{\texttt{1+2=3} \rightarrow e_1'\texttt{=3}}}{\texttt{if 1+2=3 then 2*2 else 7} \rightarrow \texttt{if } e_1'\texttt{=3 then 2*2 else 7}}$$

Using the Small-Step Semantics

$$\frac{(v_1 + v_2 = v)}{v_1 \textbf{ + } v_2 \rightarrow v}$$

$$\cfrac{\cfrac{\texttt{1+2} \rightarrow e_1'}{\texttt{1+2=3} \rightarrow e_1'\texttt{=3}}}{\texttt{if 1+2=3 then 2*2 else 7} \rightarrow \texttt{if } e_1'\texttt{=3 then 2*2 else 7}}$$

Using the Small-Step Semantics

$$\frac{(v_1 + v_2 = v)}{v_1 \mathbf{+} v_2 \to v}$$

$$\cfrac{\cfrac{\overline{\texttt{1+2} \to \texttt{3}}}{\texttt{1+2=3} \to e_1'\texttt{=3}}}{\texttt{if 1+2=3 then 2*2 else 7} \to \texttt{if}\ e_1'\texttt{=3 then 2*2 else 7}}$$

Using the Small-Step Semantics

$$\frac{(v_1 + v_2 = v)}{v_1 \textbf{ + } v_2 \rightarrow v}$$

$$\dfrac{\dfrac{}{\texttt{1+2} \rightarrow \texttt{3}}}{\dfrac{\texttt{1+2=3} \rightarrow \texttt{3=3}}{\texttt{if 1+2=3 then 2*2 else 7} \rightarrow \texttt{if 3=3 then 2*2 else 7}}}$$

Using the Small-Step Semantics

$$\dfrac{\dfrac{\overline{\texttt{1+2} \to \texttt{3}}}{\texttt{1+2=3} \to \texttt{3=3}}}{\texttt{if 1+2=3 then 2}*\texttt{2 else 7} \to \texttt{if 3=3 then 2}*\texttt{2 else 7}}$$

$$\dfrac{}{\texttt{if 3=3 then 2}*\texttt{2 else 7} \to \ \ldots}$$

- Exercise: What is the next step for the expression?

Using the Small-Step Semantics

$$\dfrac{\dfrac{\overline{\texttt{1+2} \to \texttt{3}}}{\texttt{1+2=3} \to \texttt{3=3}}}{\texttt{if 1+2=3 then 2*2 else 7} \to \texttt{if 3=3 then 2*2 else 7}}$$

$$\dfrac{}{\texttt{if 3=3 then 2*2 else 7} \to \texttt{if true then 2*2 else 7}}$$

- Exercise: What is the next step for the expression?

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Small-Step vs. Big-Step

Small-Step

- Shows intermediate states
- Allows control of evaluation order
- Closer to underlying implementation
- Can count the number of steps
- Translates directly to debugger

Big-Step

- Doesn’t need structural rules
- Clearer statement of the right results
- Sometimes internal steps don’t matter
- Translates directly to interpreter

Small-Step vs. Big-Step

Small-Step

- Shows intermediate states
- Allows control of evaluation order
- Closer to underlying implementation
- Can count the number of steps
- Translates directly to debugger

Big-Step

- Doesn't need structural rules
- Clearer statement of the right results
- Sometimes internal steps don't matter
- Translates directly to interpreter

Small-Step Interpreter/Debugger

```
let rec step (e : exp) =
```

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option =  (* e → e’ *)
```

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option =  (* e → e' *)
  match e with
  | Num i ->
  | Add (e1, e2) ->
  | …
```

Rule 0: Values are done executing!

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option =  (* e → e' *)
  match e with
  | Num i -> None
  | Add (e1, e2) ->
  | …
```

Rule 0: Values are done executing!

$$\frac{e_1 \rightarrow e_1'}{e_1 + e_2 \rightarrow e_1' + e_2}$$

$$\frac{e_2 \rightarrow e_2'}{e_1 + e_2 \rightarrow e_1 + e_2'}$$

$$\frac{(v_1 + v_2 = v)}{v_1 + v_2 \rightarrow v}$$

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option = (* e → e’ *)
  match e with
  | Add (e1, e2) -> (match e1, e2 with
     | Num v1, Num v2 -> Some (Num (v1 + v2))
```

$$\frac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$$

$$\frac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$$

$$\frac{(v_1 + v_2 = v)}{v_1 + v_2 \to v}$$

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option = (* e → e’ *)
  match e with
  | Add (e1, e2) -> (match e1, e2 with
     | Num v1, Num v2 -> Some (Num (v1 + v2))
     | Num v1, _ -> (match step e2 with
          | Some e2’ -> Some (Add (Num v1, e2’))
```

$$\frac{e_1 \rightarrow e_1'}{e_1 + e_2 \rightarrow e_1' + e_2}$$

$$\frac{e_2 \rightarrow e_2'}{e_1 + e_2 \rightarrow e_1 + e_2'}$$

$$\frac{(v_1 + v_2 = v)}{v_1 + v_2 \rightarrow v}$$

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option = (* e → e' *)
  match e with
  | Add (e1, e2) -> (match e1, e2 with
    | Num v1, Num v2 -> Some (Num (v1 + v2))
    | Num v1, _ -> (match step e2 with
      | Some e2' -> Some (Add (Num v1, e2'))
      | None -> None)
```

$$\frac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$$

$$\frac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$$

$$\frac{(v_1 + v_2 = v)}{v_1 + v_2 \to v}$$

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option = (* e → e’ *)
  match e with
  | Add (e1, e2) -> (match e1, e2 with
     | Num v1, Num v2 -> Some (Num (v1 + v2))
     | Num v1, _ -> (match step e2 with
          | Some e2’ -> Some (Add (Num v1, e2’))
          | None -> None)
     | _, _ -> (match step e1 with
```

$$\frac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$$

$$\frac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$$

$$\frac{(v_1 + v_2 = v)}{v_1 + v_2 \to v}$$

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option =  (* e → e' *)
  match e with
  | Add (e1, e2) -> (match e1, e2 with
    | Num v1, Num v2 -> Some (Num (v1 + v2))
    | Num v1, _ -> (match step e2 with
        | Some e2' -> Some (Add (Num v1, e2'))
        | None -> None)
    | _, _ -> (match step e1 with
        | Some e1' -> Some (Add (e1', e2))
        | None -> None)
```

$$\frac{e_1 \to e_1'}{e_1 + e_2 \to e_1' + e_2}$$

$$\frac{e_2 \to e_2'}{e_1 + e_2 \to e_1 + e_2'}$$

$$\frac{(v_1 + v_2 = v)}{v_1 + v_2 \to v}$$

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option =  (* e → e' *) ...

let rec debug (e : exp) : exp =  (* e → ...  → e' *)
   match step e with
   | Some e' -> debug e'
   | None -> e

eval (If (Bool true, Add (Num 3, Bool false), Num 5))
(* returns None *)
```

Small-Step Interpreter/Debugger

```
let rec step (e : exp) : exp option =  (* e → e' *) ...

let rec debug (e : exp) : exp =  (* e → ...  → e' *)
  match step e with
  | Some e' -> debug e'
  | None -> e

debug (If (Bool true, Add (Num 3, Bool false), Num 5))
(* returns Add (Num 3, Bool false) *)
```

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.